INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

DEPARTMENT OF ELECTRONICS, SYSTEMS, AND INFORMATICS

COMPUTING SYSTEMS ENGINEERING

MACHINE LEARNING COURSE

TEACHER: EDNA L. GUEVARA RIVERA

BIRD STRIKE FATALITY PREDICTION ON AIRPLANE CRASHES

DATASET CLEANING

PRESENTED BY:

MARCO RICARDO CORDERO HERNÁNDEZ, 727272

CARLOS EDUARDO RODRÍGUEZ CASTRO, 727366

OCTOBER 06TH, 2022

AUTUMN, 2022

TLAQUEPAQUE, MÉXICO

Index

Introduction

Once the project, goals and resources are defined, and as stated in the previous document, the next step toward completion it’s the *dataset cleaning*.

This process may vary depending on dataset structure, having multiple types of data crammed into several columns (referred as *features* from now on). For this particular project, additional measures had to be taken in order to transform string to numerical data.

The steps to clean the project’s dataset are described below. Just as a reminder, the dataset comes from an external source [1].

Cleaning process walkthrough

First, libraries have to be imported in order to use their methods for data loading, manipulation, and visualization.

```
# Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Next, the file containing the data itself has to be loaded. The file name it's *bird_strikes.csv*.

```
# Read dataset
ds_bst = pd.read_csv('bird_strikes.csv')
```

After the previous step, the dataset can be visualized just by invoking the store valuable.

Relevant information related to data types for each feature has to be displayed to determine which columns could be kept.

```
# Dataset info
ds_bst.info()
```

```
RangeIndex: 25558 entries, 0 to 25557
Data columns (total 26 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   record_id                              25558 non-null  int64
 1   aircraft_type                          25558 non-null  object
 2   airport_name                           25558 non-null  object
 3   altitude_bin                           25558 non-null  object
 4   aircraft_make_model                    25558 non-null  object
 5   wildlife_number_struck                 25558 non-null  object
 6   wildlife_number_struck_actual          25558 non-null  int64
 7   effect_impact_to_flight                25558 non-null  object
 8   flightdate                             25558 non-null  object
 9   effect_indicated_damage                25558 non-null  object
 10  aircraft_number_of_engines             25290 non-null  float64
 11  aircraft_airline_operator              25558 non-null  object
 12  origin_state                           25109 non-null  object
 13  when_phase_of_flight                   25558 non-null  object
 14  conditions_precipitation               25558 non-null  object
 15  remains_of_wildlife_collected          25558 non-null  bool
 16  remains_of_wildlife_sent_to_smithsonian 25558 non-null  bool
 17  remarks                                20787 non-null  object
 18  wildlife_size                          25558 non-null  object
 19  conditions_sky                         25558 non-null  object
...
 24  number_of_people_injured               25558 non-null  int64
 25  is_aircraft_large                      25558 non-null  bool
dtypes: bool(4), float64(1), int64(5), object(16)
```

Fig. 1 Dataset original features

Although output has been trimmed by the method containing library, critical information it's displayed at the bottom, indicating that 16 object type features (most likely strings) are present. Also, several boolean features are contained within other features, and although they could work in their original state, it's better to transform them into pure dichotomic values.

Before transforming present values, presence of null fields has to be taken into consideration.

```
# Null values identification
ds_bst.isna().sum(axis = 0)
```

```
record_id                                 0
aircraft_type                             0
airport_name                              0
altitude_bin                              0
aircraft_make_model                       0
wildlife_number_struck                    0
wildlife_number_struck_actual             0
effect_impact_to_flight                   0
flightdate                                0
effect_indicated_damage                   0
aircraft_number_of_engines              268
aircraft_airline_operator                 0
origin_state                            449
when_phase_of_flight                      0
conditions_precipitation                  0
remains_of_wildlife_collected             0
remains_of_wildlife_sent_to_smithsonian   0
remarks                                4771
wildlife_size                             0
conditions_sky                            0
wildlife_species                          0
pilot_warned_of_birds_or_wildlife         0
cost_total                                0
feet_above_ground                         0
number_of_people_injured                  0
is_aircraft_large                         0
```

Fig. 2 Null presence in features

Through preliminary analysis, only one of the three null value containing features will have to be transformed into full data feature, this being *aircraft_number_of_engines*, as the other ones will be suppressed later on.

```
# Null values replacement
ds_bst['aircraft_number_of_engines'].fillna(value =
int(ds_bst['aircraft_number_of_engines'].mean()), inplace = True)

print(f"Null qty remaining: {ds_bst.isna().sum(axis =
0)['aircraft_number_of_engines']}")
```

```
Null qty remaining: 0
```

Fig. 3 Null absence verification

Just as the previous step it's completed, dropping the irrelevant features will take place. These droppable features are selected by looking at the information their data holds. Remaining features are shown below the step code.

```
# Non-relevant columns dropping
nrc = [
    'record_id',
    'airport_name',
    'wildlife_number_struck',
    'flightdate',
    'aircraft_airline_operator',
    'origin_state',
    'remains_of_wildlife_sent_to_smithsonian',
    'remarks',
    'wildlife_species',
    'cost_total'
]
'''
    Although 'cost_total' could be used, the information related to
that feature
    it's only obtained after the accident has occurred
'''

ds_bst.drop(nrc, inplace = True, axis = 1)
ds_bst.info()
```

```
Data columns (total 16 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   aircraft_type                    25558 non-null  object
 1   altitude_bin                     25558 non-null  object
 2   aircraft_make_model              25558 non-null  object
 3   wildlife_number_struck_actual    25558 non-null  int64
 4   effect_impact_to_flight          25558 non-null  object
 5   effect_indicated_damage          25558 non-null  object
 6   aircraft_number_of_engines       25558 non-null  float64
 7   when_phase_of_flight             25558 non-null  object
 8   conditions_precipitation         25558 non-null  object
 9   remains_of_wildlife_collected    25558 non-null  bool
 10  wildlife_size                    25558 non-null  object
 11  conditions_sky                   25558 non-null  object
 12  pilot_warned_of_birds_or_wildlife 25558 non-null  bool
 13  feet_above_ground                25558 non-null  int64
 14  number_of_people_injured         25558 non-null  int64
 15  is_aircraft_large                25558 non-null  bool
dtypes: bool(3), float64(1), int64(3), object(9)
```

Fig. 4 Remaining features after deletion

Now, just as stated before, object/string and bool features have to be transformed to numerical values. Two functions were made, one to transform categorical values, and another to turn boolean values to their binary representation.

```python
def categorize(dataset, feature):
    holder = {}
    index = 0

    for row in dataset[feature]:
        if (row not in holder):
            holder[row] = index
            index += 1

    for val in holder:
        dataset[feature] = dataset[feature].replace([f'{val}'], holder[val])

def to_binary(dataset, feature):
    dataset[feature] = dataset[feature].apply(lambda x : 1 if x else 0)
```

With the aid of Fig. 4, indexes of each feature and their corresponding transformation can be done easily.

```
features = ds_bst.columns.values
to_modify = (0, 1, 2, 4, 5, 7, 8, 10, 11)
to_bin = (9, 12, 15)

# Implementation not recommended for long features lenght (<50)
for i in range(16):
    if (i in to_modify):
        categorize(ds_bst, features[i])
    elif (i in to_bin):
        to_binary(ds_bst, features[i])

ds_bst.info()
```

```
Data columns (total 16 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   aircraft_type                    25558 non-null  int64
 1   altitude_bin                     25558 non-null  int64
 2   aircraft_make_model              25558 non-null  int64
 3   wildlife_number_struck_actual    25558 non-null  int64
 4   effect_impact_to_flight          25558 non-null  int64
 5   effect_indicated_damage          25558 non-null  int64
 6   aircraft_number_of_engines       25558 non-null  float64
 7   when_phase_of_flight             25558 non-null  int64
 8   conditions_precipitation         25558 non-null  int64
 9   remains_of_wildlife_collected    25558 non-null  int64
 10  wildlife_size                    25558 non-null  int64
 11  conditions_sky                   25558 non-null  int64
 12  pilot_warned_of_birds_or_wildlife 25558 non-null  int64
 13  feet_above_ground                25558 non-null  int64
 14  number_of_people_injured         25558 non-null  int64
 15  is_aircraft_large                25558 non-null  int64
dtypes: float64(1), int64(15)
```

Fig. 5 Transformed remaining features

Heading towards end of cleaning process, a visualization approach has to be taken in order to detect repeated values.

```
ds_bst.hist(bins = 30, figsize = (20, 20), color = 'r')
```

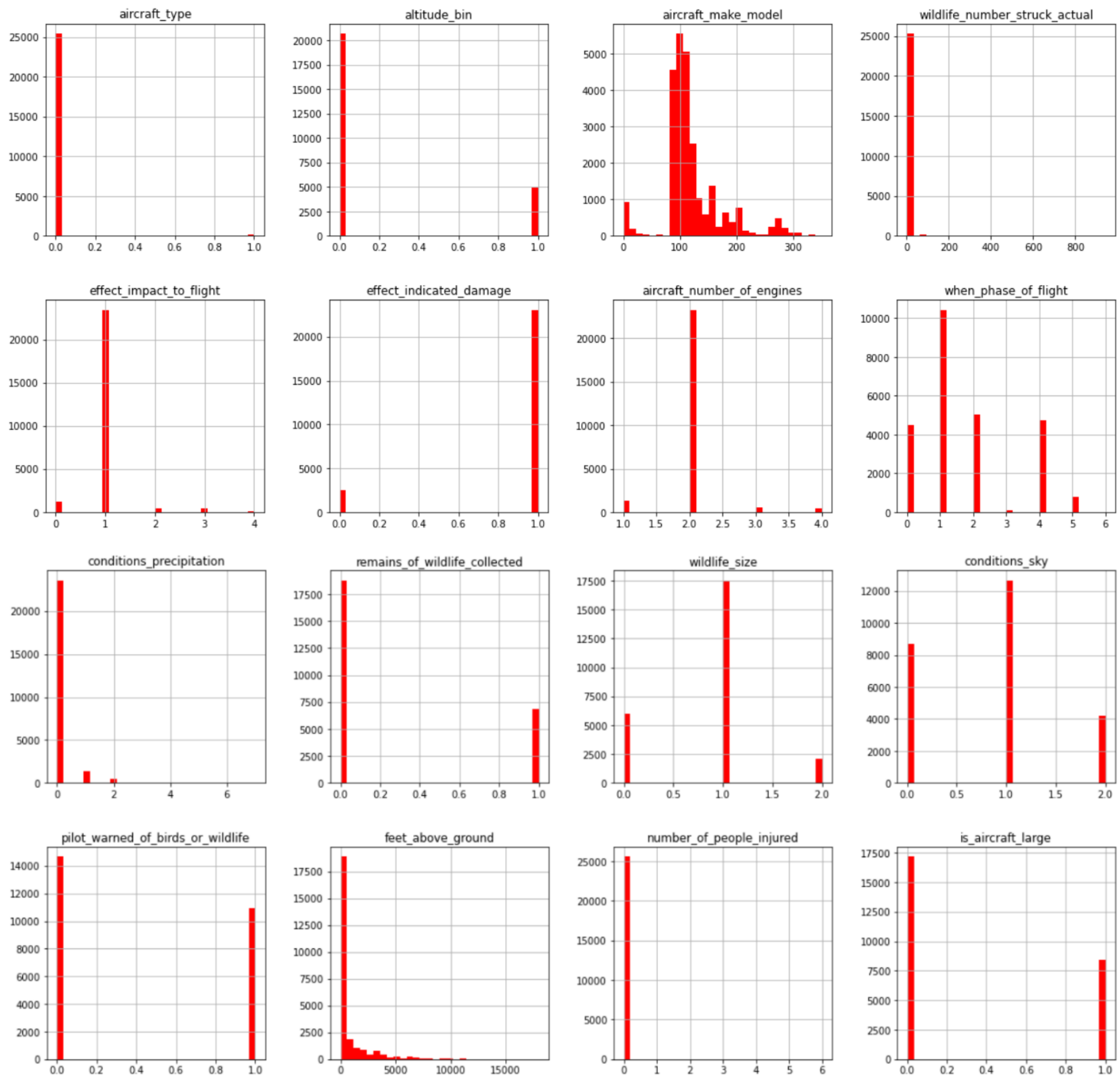

Fig. 6 Histograms

This extra analysis was helpful, because it ultimately helped in the exclusion of another two useless features.

```
ds_bst.drop(['aircraft_type', 'number_of_people_injured'], inplace =
True, axis = 1)
```

With that last step, the process of dataset cleaning has concluded. Additional analysis and clean dataset file generation can be found inside the jupyter notebook file.

Conclusions and pending work

Now that the cleaning of the dataset has been made, the next step regarding the project would be the application of ML algorithms to predict an outcome given the relevant features.

References

[1] J. Shih, «data.world,» 2016. [Online]. Available: https://data.world/shihzy/2000-2011-birds-strikes-planes. [Last access: 02 October 2022].